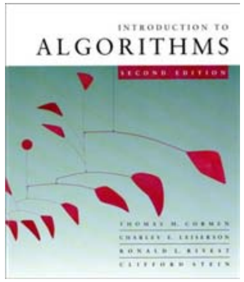
CS 5633 -- Spring 2005



B-trees II

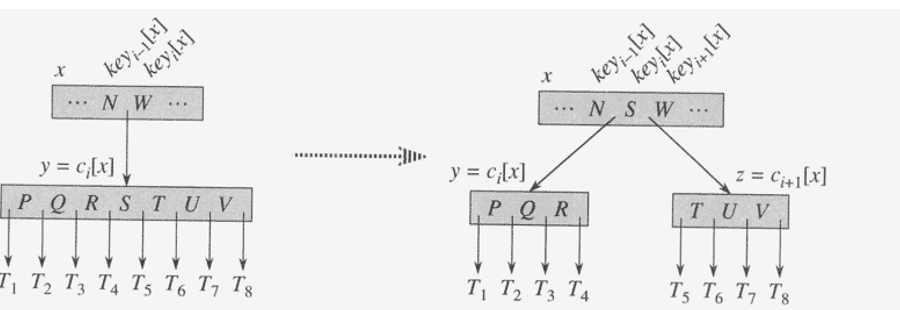
B-tree insert

- There are different insertion strategies. We just cover one of them
- Make one pass down the tree:
  - The goal is to insert the new key *key* into a leaf
  - Search where *key* should be inserted
  - Only descend into non-full nodes:
    - If a node is full, split it. Then continue descending.
    - Splitting of the root node is the only way a B-tree grows in height

B-TREE-SPLIT-CHILD(x,i,y)

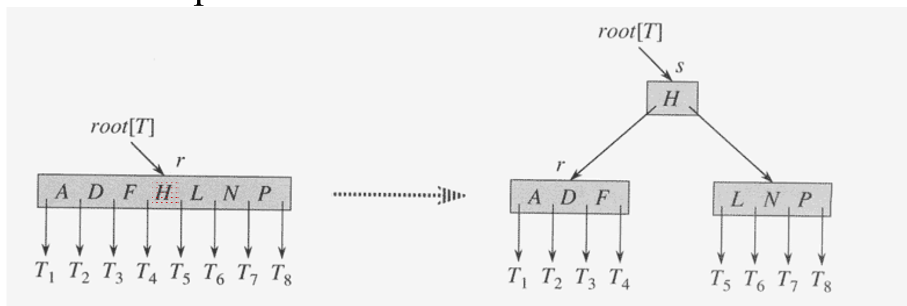
has $2k$-1 keys

- Split full node y into two nodes y and z of k keys
- Median key S of y is moved up into y's parent x
- Example below for k = 4



Split root: B-TREE-SPLIT-CHILD(s,1,r)

- The **full** root node r is split in two.
- A new root node s is created
- s contains the median key H of r and has the two halves of r as children
- Example below for k = 4



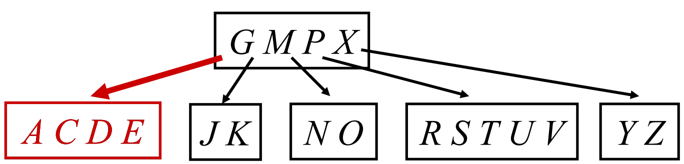
B-Tree-Insert(*T*,*key*)

```
r = root[T]
if (# keys in r) = 2k-1 // root r is full
    //insert new root node:
    s ← ALLOCATE-NODE()
    root[T] ← s
    // split old root r to be two children of new root s
    B-TREE-SPLIT-CHILD(s,1,r)
    B-TREE-INSERT-NONFULL(s,key)
else B-TREE-INSERT-NONFULL(s,key)
```

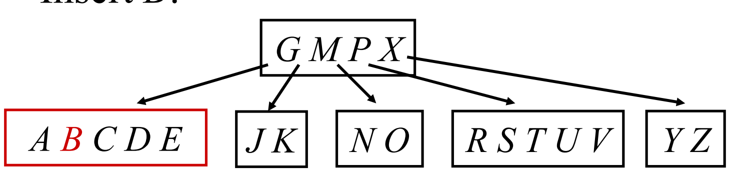

B-Tree-Insert-Nonfull(*x*,*key*)

```
if x is a leaf then
    insert key at the correct (sorted) position in x
    DISK-WRITE(x)
else
    find child c of x which by the search tree property
      should contain key
    DISK-READ(c)
    if c is full then  // c contains 2k-1 keys
        B-TREE-SPLIT-CHILD(x,i,c)
        B-TREE-INSERT-NONFULL(c,k)
```

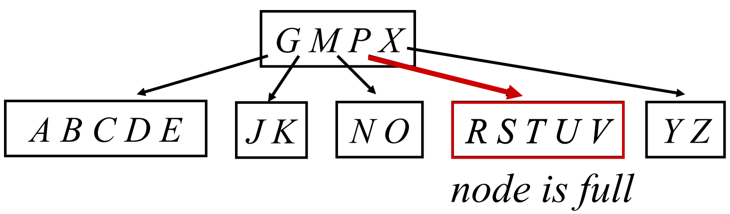

Insert example (k=3)



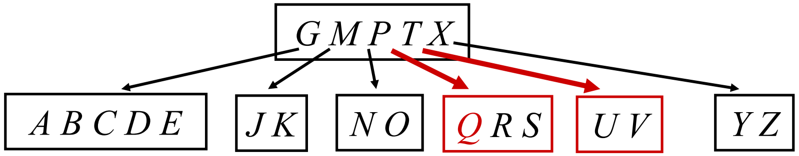
- Insert *B*:



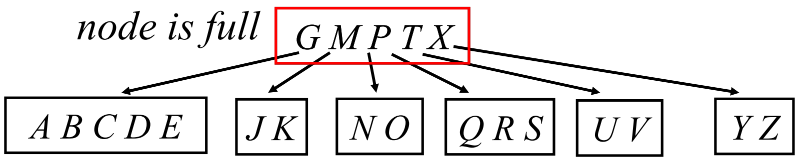
Insert example (k=3) -- cont.



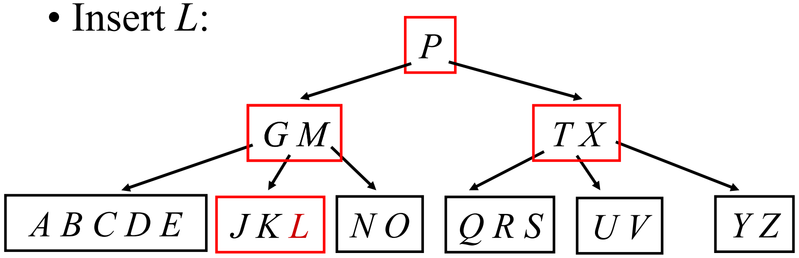
- Insert *Q*:



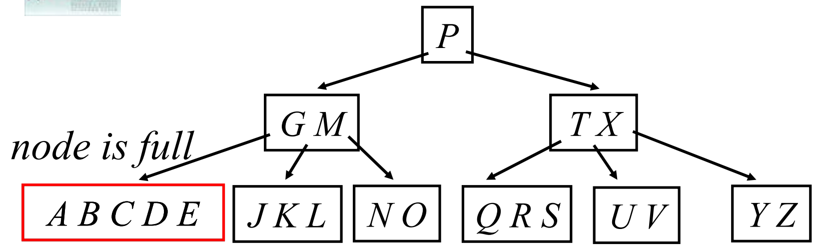
Insert example (k=3) -- cont.



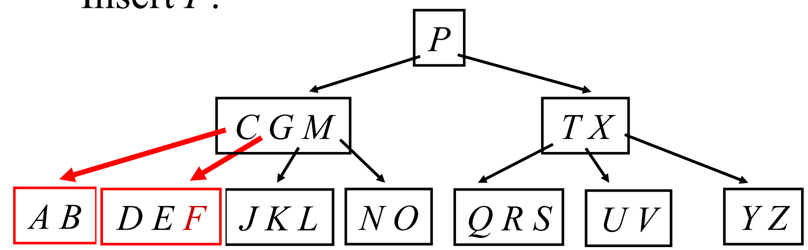
- Insert L:



Insert example (k=3) -- cont.



- Insert F:



Runtime of B-TREE-INSERT

- $O(k)$ runtime per node
- Path has height $h = O(\log_k n)$
- CPU-time: $O(k \log_k n)$

- Disk accesses: $O(\log_k n)$

  disk accesses are more expensive than CPU time

B-trees -- Conclusion

- B-trees are balanced k-ary search trees
- The **degree** of each node is **bounded from above and below** using the parameter k
- All leaves are at the same height
- **No rotations** are needed: During insertion (or deletion) the balance is maintained by node **splitting** (or node **merging**)
- The tree grows (shrinks) in height only by splitting (or merging) the root